The umbral transfer-matrix method
IV. Counting self-avoiding polygons and walks

Doron ZEILBERGER [1]
<zeilberg@math.temple.edu>
Submitted: March 13, 2001; Accepted: July 26, 2001.
MR Subject Classifications: 05A

To think in a computerized way is an important matter ... to go with it to the end of the limit of possibilities, and there to develop new, unpredictable ones. — David Avidan

(Free translation of an excerpt from the Hebrew original of Avidan's poem *lakhshov betsura memukhshevet* [To Think In a Computerized Way].)

Abstract: This is the fourth installment of the five-part saga on the Umbral Transfer-Matrix method, based on Gian-Carlo Rota's seminal notion of the umbra. In this article we describe the Maple packages `USAP`, `USAW`, and `MAYLIS`. `USAP` automatically constructs, for any specific r, an Umbral Scheme for enumerating, according to perimeter, the number of self-avoiding polygons with $\leq 2r$ horizontal edges per vertical cross-section. The much more complicated `USAW` does the analogous thing for self-avoiding walks. Such Umbral Schemes enable counting these classes of self-avoiding polygons and walks in polynomial time as opposed to the exponential time that is required by naive counting. Finally `MAYLIS` is targeted to the special case of enumerating classes of saps with at most two horizontal edges per vertical cross-section (equivalently column-convex polyominoes by perimeter), and related classes. In this computationally trivial case we can actually *automatically* solve the equations that were *automatically* generated by `USAP`. As an example, we give the first *fully computer-generated* proof of the celebrated Delest-Viennot result that the number of convex polyominoes with perimeter $2n+8$ equals $(2n+11)4^n - 4(2n+1)!/n!^2$.

The Third (and Ultimately Most EFFICIENT) Way of Using MAPLE

The great combinatorial enumerator, Mireille Bousquet-Mélou, in her fascinating *Rapport* ([B], p. 20), writes about the *two* ways she uses Maple. The first, more traditional one, is *en aval* (downstream, i.e. *after* the research), which consists of verifying or correcting an already proven identity. The second, *beaucoup plus enthousiasmante*, is *en amont* (upstream, i.e. *during* the research). And indeed Bousquet-Mélou, and the other members of the celebrated *école bordelaise* (Xavier Viennot, Maylis Delest, and their academic descendants), are real whizzes in this *interactive* mode of research.

But, there is yet *another* way of using Maple, which is my own personal favorite, and that is exemplified in the project described in this article. This, *third* way, of using

[1] Department of Mathematics, Temple University, Philadelphia, PA 19122, USA.
<http://www.math.temple.edu/~zeilberg/>. Accompanied by the Maple packages `USAP`, `USAW` and `MAYLIS`, all of which are available either from the above home page of Zeilberger, or from <http://www.math.temple.edu/~zeilberg/utm.html>. Supported in part by the NSF.

Maple, is to let the *computer do everything all by itself*. The human's role in such an endeavor is two-fold. First one has to invent a fruitful *concept*, that may be turned into an algorithm. This part was done, in the present case, by Gian-Carlo Rota (see [Z1]) who invented the *umbra*. Then the task remains to design an algorithm, and write a program implementing it, that will guide the computer to 'do research'. Of course, once the computer knows what to do, it can go much further than any human.

So this is neither down- nor up- stream, but *above*-stream. We take a general problem, say, of enumerating certain classes of self-avoiding polygons, and use Maple, not just to *solve* the humanly-derived equations, that we "cleverly" found by doing *combinatorial reasoning*, but let the computer do everything! (except, at this time of writing, the programming). This consists of having the computer first *derive the equations*, and then, whenever, possible, *solve them*. To take a dramatic example, the Delest-Viennot [DV] result, mentioned in the abstract, can now be proved in less than 15 seconds of CPU time. Just type, in `MAYLIS`: `ProveDelestViennot();`. Moreover, for the same effort of writing a program to find a *specific* generating function, we can write a much more general program, and get thousands of new results. Also, our much faster machine colleagues, can easily surpass us. For example, the computer-generated *set of equations* for enumerating saps with at most 4 horizontal edges per vertical cross-section, occupies seven pages (see the appendix), and it probably can't be "solved" in any *reasonable* way, but, as was pointed out by Herb Wilf, an "answer" is just an algorithm, and it is a "good answer" if the algorithm is fast (or at least polynomial-time), and from this perspective, the *set of equations* says it all.

Granted, even computers have their limits, and, for example, my computer, Shalosh B. Ekhad, ran out of memory when it tried to find the set of equations for the next-in-line case of finding an Umbral Scheme for enumerating self-avoiding polygons with at most six horizontal edges per vertical cross-section. But, who cares about output? It would not be humanly-readable in any case, since it would probably occupy more than a hundred pages. The *program* to find the equations can be enjoyed for its own sake, so an even more enlightened definition of *answer* is the *computer program* that would find the answer if you had a sufficiently large and fast computer.

I believe that most of us humans would do well to retire from proving, and take up programming. Of course, we still need a handful of humans, of the caliber of *prophets* like Gian Carlo-Rota, to invent new *concepts*, that could be turned into computer programs, but the day-to-day activity of *proving*, even computer-assisted, should, and would, become *passé*.

What makes this third way of using Maple so hard, at present, is that Maple is really geared to be used interactively. We need higher-and-higher level programming languages, that would make the Maple packages described in this article look like assembly-language code. Hence, perhaps the most important potential impact of this modest effort is as a guide for these future super-Maple designers.

Required Reading

The reader is expected to be familiar with [Z1]. It would also help to read the parts of [$Z0$] concerned with finding generating functions for counting self-avoiding polygons and walks with vertical cross-sections of bounded width. The present treatment is inspired by the finite case described in detail in [Z0], but with the Umbral twist, getting matrices whose entries are *operators* rather than mere polynomials, as *transfer matrices*.

The Alphabet For Self-Avoiding Polygons

A *self-avoiding polygon* (henceforth sap) on the square lattice is a finite connected induced subgraph of the square lattice with every vertex having degree 2. Of course we identify two saps that are translations of each other. From now on, we will take the smallest x coordinate of any of the lattice points participating in the sap, to be 0. We may also take the smallest y coordinate to be 0, but this is irrelevant to our approach.

We can "read" a sap from left to right, by considering, for $k = 0, 1, 2, \ldots$, the edges that belong to the vertical cross-sections $k \le x < k + 1$. There are two kinds of edges: vertical, and horizontal.

Consider the horizontal edges, i.e. edges joining (k, y) and $(k + 1, y)$, for some y. It is immediate that there are always an even number of such horizontal edges.

Here we will undertake the task of enumerating saps with at most $2r$ such horizontal edges (per vertical cross-section), where r is prescribed in advance. In particular the case $r = 1$ is the much studied case of *column-convex* polyominoes according to perimeter.

Note that these horizontal edges arrange naturally into pairs that are "reachable from one side", i.e. one of the two portions of the sap that are between them is entirely to the left of $x = k$, (and hence the other portion is entirely to the right of $x = k$). For each such pair of edges, we will assign the letter L to the bottom one, and the letter R to the top one. It is immediate that the resulting word is a legal parentheses (a.k.a. Dyck word) where L stands for "(" and R stands for ")". Recall that a *legal parentheses* is a word in $\{L, R\}$ with as many as L's as R's, and such that any prefix has at least as many L's as R's. Conversely, every such Dyck word may show up, eventually, in some sap.

As we said above, we are interested in enumerating, according to the perimeter, for any fixed r, the subfamily of saps that have at most r $L - R$ pairs in every vertical cross-section. When $r = 1$, we have the so-called vertically convex animals according to perimeter. Unlike [Z0], where we also restricted the width of the cross-sections, and hence always obtained *rational* generating functions (because of the finite Markovity), now we allow arbitrary width, and the transfer matrices will contain Rota operators rather than mere polynomials.

The size of the alphabet for saps with at most r $L - R$ pairs is $C_1 + C_2 + \ldots + C_r$, where $C_i = \binom{2i}{i}/(2i + 1)$ are the Catalan numbers. For example, where $r = 1$, we only have one letter: LR, while when $r = 2$, the alphabet is $\{LR, LRLR, LLRR\}$, when $r = 3$ it is

$$\{LR, LRLR, LLRR, LRLRLR, LRLLRR, LLRRLR, LLRLRR, LLLRRR\} \quad \text{, etc.}$$

The Umbral Letters for SAPs

However, each of these letters is really a parameterized family of letters. For a letter with s L-R pairs ($1 \leq s \leq r$), there are $2s-1$ gaps between the edges, and we will denote the generic lengths of these gaps by $A_1, A_2, \ldots, A_{2s-1}$. The corresponding x-weight of such a letter is the generic monomial $x_1^{A_1} x_2^{A_2} \ldots x_{2s-1}^{A_{2s-1}}$. Note that the x's are but *catalytic variables*, and we will soon also introduce q to keep track of the perimeter.

The Leftmost Letters For SAPs

In addition to the above-introduced genuine letters, it will be convenient to introduce two extra letters: START, and END, depending on no x-variables (i.e. they only depend on q).

The first letter (right after the fictitious START) may be LR, or $LRLR$, ..., up to $(LR)^r$, i.e. $(LR)^s$, for $s = 1, \ldots, r$. Since for the leftmost vertical cross-section $0 \leq x < 1$, the only way that two paired edges can reach each other from the left is via the vertical line $x = 0$, hence every such LR pair must be adjacent.

Now these leftmost letters can be of arbitrary size. If the immediate follower of START is $(LR)^s$ then its generic x- weight is $x_1^{A_1} \ldots x_{2s-1}^{A_{2s-1}}$. But the first L is connected (on $x = 0$) to the first R (contributing a stretch of A_1 to the perimeter), the second L is connected to the second R (contributing A_3 to the perimeter), etc. . In addition, the $2s$ horizontal edges joining $x = 0$ and $x = 1$ contribute $2s$ to the perimeter. Hence, we have that the pre-umbra acting on START is:

$$Z[START] \rightarrow$$

$$q^2 \frac{qx_1}{1-qx_1} Z[LR] + q^4 \frac{qx_1}{1-qx_1} \frac{x_2}{1-x_2} \frac{qx_3}{1-qx_3} Z[LRLR] + \ldots +$$
$$q^{2r} \frac{qx_1}{1-qx_1} \frac{x_2}{1-x_2} \frac{qx_3}{1-qx_3} \cdots \frac{x_{2r-2}}{1-x_{2r-2}} \frac{qx_{2r-1}}{1-qx_{2r-1}} Z[(LR)^r] \quad .$$

There Are Many Ways to Continue to the Next Vertical Cross-Section

The continuation from one vertical cross-section, say $k-1 \leq x < k$, to the next, $k \leq x < k+1$ takes place in three phases.

Phase I: The PrePreFollowers

Suppose that the pattern of the horizontal edges in the vertical cross-section $k-1 \leq x < k$ is a certain legal $L-R$ word. It may be continued in many ways. The first phase is to decide how to unite some of L's and R's by joining them on $x = k$, thereby creating new vertical edges. We will denote by C these "closed-up" former L's and R's.

There are three legal moves:

(i) JoinLL, obtained by joining two adjacent L's, making them both C's, and changing the R-mate of the upper L into an L, thereby preserving the balance of L's and R's.

For example, there is only one way to apply a JoinLL operation to the SAP-Umbral letter LLRR, turning it into CCLR. For LLLRRR there are two legal JoinLL operations, one obtained by joining the first 2 L's, turning LLLRRR into CCLRLR, and the other obtained by joining the second and third L, getting LCCLRR. We may apply as many JoinLL operations as we want, as long as they don't interfere with each other.

(ii) JoinRR, obtained by joining two adjacent R's, making them both C's, and changing the L-mate of the lower R into an R, hence preserving the balance of L's and R's.

For example, there is only one way to apply a JoinRR operation to the SAP-Umbral letter LLRR, turning it into LRCC. For LLLRRR there are two legal JoinRR operations, one obtained by joining the first 2 R's, turning LLLRRR into LLRCCR and the other obtained by joining the second and third R, getting LRLRCC. We may apply as many JoinRR operations as we want, as long as they don't interfere with each other.

(iii) JoinRL, obtained by joining an R to an L immediately above it, and changing them both to C's. This does not change any of the other L's or R's, but the widowers of the deceased R and L now are "married" to each other. For example, there is only one way to perform a JoinRL operation to the SAP letter LRLR, resulting in LCCR, while for LRLRLR we may get LCCRLR and LRLCCR. If we operate on both adjacent RL pairs, we get LCCCCR.

To get all the PrePreFollowers of a given SAP-letter, we apply JoinLL, JoinRR, and JoinRL in any conceivable order, except that, right now, we want to leave at least one LR pair, and not turn them all to C's.

For example, the set of PrePreFollowers of LRLLRR is

$$\{LRCCLR, LRLRCC, LCCLRR\}.$$

For future reference, we also need to record those portions of $x = k$ that have been "closed to traffic", because of the above joining operations. If the input SAP-letter (i.e. the one whose continuations we are investigating) has s LR pairs, and hence $2s$ L's and R's combined, let's number them by the integers 1 through $2s$, and denote the resulting consecutive intervals by $\{[0,1],[1,2],[2,3],\ldots,[2s-1,2s],[2s,2s+1]\}$. Here $[0,1]$ and $[2s,2s+1]$ are the "infinite" intervals below the bottom L and above the top R, respectively. Now some of these intervals are currently closed for business because of the joining operation, and we need to record it. The full PrePreFollower also records this information. Hence the set of PrePreFollowers of LRLLRR are:

$$\{[LRCCLR,\{[3,4]\}],[LRLRCC,\{[5,6]\}],[LCCLRR,\{[2,3]\}]\} \quad .$$

I apologize for the redundant information, but when one programs, it is often convenient to have data structures with redundancy.

Phase II: The PreFollowers

The next decision is how to continue the surviving L's and R's from $x < k$ into $k \leq x < k+1$. Each of the L's and R's that survived Phase I has to decide, independently, whether to cross the $x = k$ road straight away (we will call this option 0), or to walk some distance up $x=k$ before crossing (option 1), or to walk some distance down $x=k$ before crossing to $x>k$ (option -1). If a PrePreFollower has t LR-pairs left (all the other L's and R's having turned into C's) then there are 3^{2t} possibilities altogether.

We will denote each PreFollower by a list each of whose elements is either C, or $[L,-1]$, or $[L,0]$, or $[L,1]$, or $[R,-1]$, or $[R,0]$, or $[R,1]$. For example, the PrePreFollower $[CCLR, \{[1,2]\}]$ of LLRR gives rise to the following $9=3^2$ PreFollowers:

$$\{[[C,C,[L,-1],[R,-1]],\{[1,2]\}],[[C,C,[L,-1],[R,0]],\\ \{[1,2]\}],[[C,C,[L,-1],[R,1]],\{[1,2]\}],$$

$$[[C,C,[L,0],[R,-1]],\{[1,2]\}],[[C,C,[L,0],[R,0]],\{[1,2]\}],[[C,C,[L,0],[R,1]],\{[1,2]\}],$$

$$[[C,C,[L,1],[R,-1]],\{[1,2]\}],[[C,C,[L,1],[R,0]],\{[1,2]\}],[[C,C,[L,1],[R,1]],\{[1,2]\}]\}.$$

Phase III: The Followers

Every vertical cross-section $k \le x < k+1$ is allowed up to r LR pairs. If the examined PreFollower has less, then the sap may decide to insert new LR pairs, in the remaining open slots, as long as the total number does not exceed the maximal allowed number, r. Of course, these new pairs behave like the ones at the very beginning, they come in *adjacent* LR pairs. Also they may only be inserted in the "open space" intervals, which consists of the complement of the second component of the PreFollower with respect to the set of all available intervals

$$\{[0,1],[1,2],\ldots,[2s-1,2s],[2s,2s+1]\} \quad .$$

Recall that a PreFollower is a list of length two. A Follower will be a list of length four. To construct the set of Followers stemming from a given PreFollower, we retain the two components of the PreFollower. To this we append a third component: the list of available open intervals (listed in the natural, increasing order), followed by another list that codes our decision where to insert new LR pairs, as described below.

Consider, for example, the following PreFollower of LLRR:

$$[[C,C,[L,1],[R,1]],\{[1,2]\}]$$

(obtained by performing a JoinLL operation). The totality of intervals of the source-SAP-letter, LLRR, is the set

$$\{[0,1],[1,2],[2,3],[3,4],[4,5]\} \quad ,$$

and removing the "closed for traffic" interval [1,2], gives us, as the set of available intervals:

$$[[0,1],[2,3],[3,4],[4,5]],$$

but, we store it as a list rather than a set, so that we can refer to its entries conveniently.

Since we are doing the language of $SAP(r)$, i.e. saps with at most r LR-pairs for each vertical cross-section, and the PreFollower in question has t LR-pairs, this means that we may insert up to $r-t$ (adjacent!) LR pairs in the open intervals. We denote this choice by a list of integers $[a_1, a_2, \ldots, a_m]$, where m is the number of intervals open to traffic (the length of the above-mentioned third component of the current Follower), and the meaning is that we inserted a_1 new adjacent LR pairs in the first available open-interval, a_2 in the second, and so on. Of course we must have $a_1 + a_2 + \ldots + a_m \leq r - t$.

Thus, if $r = 3$, the PreFollower

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}]$$

of LLRR, gives rise to the following Followers, among many others

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [2, 0, 0, 0]] \quad ,$$

where we decided to place two new adjacent LR pairs at the semi-infinite bottom interval, or

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [1, 1, 0, 0]] \quad ,$$

where we decided to put one new adjacent LR pair at the semi-infinite bottom interval, and one in the interval between the second C and the L, or

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 0, 2]] \quad ,$$

where we decided to place two new adjacent LR pairs at the semi-infinite top interval, and so on. We may also choose not to insert any new LR pairs (since we are allowing a vertical cross-section to have less than the maximum allotment of horizontal edges), hence the following PreFollower is also OK:

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 0, 0]] \quad ,$$

as is

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 1, 0]] \quad ,$$

where we inserted only one new LR pair between the L and the R of the PreFollower.

The total number of Followers stemming from the above PreFollower (still with $r = 3$) is the number of vectors of non-negative integers, $[a_1, a_2, a_3, a_4]$ with $a_1 + a_2 + a_3 + a_4 \leq 2$.

The Emerged Letter

It soon becomes very clear that there are lots of ways to continue a given SAP-letter from one vertical cross-section to the next, but many different continuations will produce the same SAP-letter in the next vertical cross-section. To find the induced letter, simply ignore the C's, and take note of the newly inserted LR pairs. Hence the following Follower of LLRR

$$[[C, C, [L, 1], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 1, 0]] \quad ,$$

gives rise to the letter LLRR, while

$$[[C,C,[L,1],[R,1]],\{[1,2]\},[[0,1],[2,3],[3,4],[4,5]],[0,0,0,2]] \quad ,$$

gives rise to LRLRLR, etc.

A Very Important Polynomial In This Algorithm

Let A be a *symbol* denoting an integer, and let $z_1, z_2, \ldots, z_m$ be variables, we let

$$P_A(z_1,\ldots,z_m) = \sum z_1^{i_1} z_2^{i_2} \cdots z_m^{i_m} \quad ,$$

where the sum extends over all m-tuples of *positive* integers $(i_1,\ldots,i_m)$ satisfying $i_1 + \ldots + i_m = A$. Note that $P_A(z_1,\ldots,z_m) = z_1 \cdots z_m h_{A-m}(z_1,\ldots,z_m)$, where h_i is the usual *complete homogeneous symmetric polynomial* of degree i.

If $Z = \{z_1,\ldots,z_m\}$ is a set of variables, we will sometimes write the above as $P_A(Z)$, which is legitimate, since P_A is a symmetric function of its arguments.

We have, of course,

$$P_A(z_1,\ldots,z_m) = \sum_{i=1}^{A-1} P_{A-i}(z_1,\ldots,z_{m-1}) z_m^i \quad .$$

Since $P_A(z_1) = z_1^A$, we can repeatedly use this inductive formula to get $P_A(z_1,\ldots,z_m)$ for any m. All that is involved is summing geometric series, that Maple does very well. For example

$$P_A(z_1,z_2) = \frac{z_1^A z_2 - z_1 z_2^A}{z_1 - z_2} \quad .$$

Note that the evaluated expression of $P_A(z_1,\ldots,z_m)$ is a linear combination of $z_1^A, z_2^A, \ldots, z_m^A$ with coefficients that are rational functions of $z_1, z_2, \ldots, z_m$.

The Umbral Evolution

Each SAP-letter with s L's and s R's, $(1 \leq s \leq r)$ is associated with the generic monomial

$$x_1^{A_1} x_2^{A_2} \ldots x_{2s-1}^{A_{2s-1}} \quad ,$$

where the generic positive integers $A_1, A_2, \ldots, A_{2s-1}$ denote the lengths of the intervals between consecutive horizontal edges corresponding to the entries of the SAP-letter.

For any of the (many) Followers of a letter, it is possible to predict the totality (i.e. the generating function) of the contribution to the (q,x) weight coming from the transition under discussion. For any given SAP-letter LET and any given Follower $PreLET$, we denote by $APU(LET, PreLET)$ this generating function. APU stands for *Atomic Pre-Umbra.* How to find it?

First, we treat the special case where all the second components of the L's and R's of the first component are 0, i.e. all the L's and R's that survived the joining process of phase I, decide not to dawdle by wondering vertically on $x = k$, but rather walk the inevitable horizontal edge straight away. Then we insert the new LR pairs as prescribed by the fourth component of the Follower. Now we number the resulting new intervals (of the emerged SAP-letter), by $0, 1, \ldots, 2m-1, 2m$, assuming that there are m L's and m R's in the emerging successor SAP-letter.

Next we look how the $2s+1$ intervals of the original SAP-letter interface with the $2m+1$ intervals of its successor. Some of the intervals of the successor SAP-letter are those coming from newly inserted LR pairs, hence contribute also to the q-weight. If the i^{th} interval (of length A_i) of the originating letter, overlaps with intervals j_i through $j_i + p_i$ (for some $p_i \geq 0$), of the successor letter, and we set $a_{i,s} = 1$ or $a_{i,s} = 0$ according to whether the interval $j_i + s$ is due to a newly-inserted LR-pair, ($s = 0, 1, \ldots, p_i$), or not, respectively, then the new contribution to the weight, only stemming from that A_i is

$$P_{A_i}(q^{a_{i,0}} x_{j_i}, q^{a_{i,1}} x_{j_i+1}, \ldots, q^{a_{i,p_i}} x_{j_i+p_i}) \quad .$$

Note that $a_{i,0}$ and a_{i,p_i} must always be zero. In addition, to take care of the q-part of the weight (that keeps track of the perimeter, our primary concern), we multiply this by q^{2m} (for the $2m$ horizontal edges of the successor SAP-letter), and by $q^{A_{i_1}+A_{i_2}+\ldots+A_{i_v}}$, if the "closed for traffic" intervals were the intervals $i_1, i_2, \ldots, i_v$.

Summarizing, the atomic pre-Umbra, in this (no vertical dawdling) case, is

$$q^{A_{i_1}+A_{i_2}+\ldots+A_{i_v}+2m} \prod_{i=0}^{2s+1} P_{A_i}(q^{a_{i,0}} x_{j_i}, q^{a_{i,1}} x_{j_i+1}, \ldots, q^{a_{i,p_i}} x_{j_i+p_i}) \quad . \qquad (Monster)$$

Here we set $A_0 = A_{2s+1} = \infty$. Also later we will remove x_0 from the argument of P_{A_0}, remove x_{2m} from the argument of $P_{A_{2s+1}}$, and then set x_0 and x_{2m} to 1 (since they contribute to the bottom and top infinite intervals, that do not have x-weight contribution).

For example,

$$APU(LLRR, [[C, C, [L, 0], [R, 0]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 1, 0]])$$

is the pre-Umbra

$$x_1^{A_1} x_2^{A_2} x_3^{A_3} \rightarrow q^{A_1+4} P_{A_0}(x_0) P_{A_1}(x_0) P_{A_2}(x_0) P_{A_3}(x_1, q x_2, x_3) P_{A_4}(x_4) \quad .$$

Now we are ready to treat the Atomic Pre-Umbra coming from the case where some (or all) the L's and R's of the PreFollower decide to wonder up or down on $x = k$ before venturing into $x > k$. Suppose that the horizontal edge consisting of the bottom of the interval whose length is A_i is an L or R, that goes down (denoted by [L,-1], or [R,-1]) in the first component of the PreFollower. This means that the interval (of the source-LETTER) immediately below it (of length A_{i-1}) gets to overlap A_i's bottom-interfaced successor SAP-letter interval, j_i. In other words, A_i is generous and shares its bottom interfaced interval with its neighbor below. Not only that, A_{i-1} gets q-credit. So, if this is the case, in $(Monster)$, we have to replace

$$P_{A_{i-1}}(x_{j_{i-1}}, q^{a_{i-1,1}}x_{j_{i-1}+1}, \ldots, q^{a_{i-1,p_{i-1}-1}}x_{j_{i-1}+p_{i-1}-1}, x_{j_{i-1}+p_{i-1}})$$

by

$$P_{A_{i-1}}(x_{j_{i-1}}, q^{a_{i-1,1}}x_{j_{i-1}+1}, \ldots, q^{a_{i-1,p_{i-1}-1}}x_{j_{i-1}+p_{i-1}-1}, x_{j_{i-1}+p_{i-1}}, qx_{j_i}) \quad ,$$

(of course $j_i = j_{i-1} + p_{i-1} + 1$).

Similarly, suppose that the entry corresponding to the horizontal edge consisting of the top of the source-letter interval whose length is A_i is an L or R, that goes up (denoted by [L,1] or [R,1]) in the PreFollower. This means that the interval right below it (of length A_{i-1}) is willing to share its top successor-letter interval, $j_{i-1} + p_{i-1}$. In other words, A_{i-1} is generous and shares its top interfaced interval with its neighbor above. Not only that, A_i gets q-credit. So if this is the case, in $(Monster)$, we have to replace

$$P_{A_i}(x_{j_i}, q^{a_{i,1}}x_{j_i+1}, \ldots, x_{j_i+p_i})$$

by

$$P_{A_i}(qx_{j_{i-1}+p_{i-1}}, x_{j_i}, q^{a_{i,1}}x_{j_i+1}, \ldots, x_{j_i+p_i}) \quad .$$

We do these adjustments for each and every time an L or R is not attached to 0. It is easy to see that these adjustments are disjoint, i.e. do not interfere with each other, and can be carried in any order. The final outcome is the Atomic Pre-Umbra corresponding to this particular SAP-letter and particular Follower.

For example,

$$APU(LLRR, [[C, C, [L, 0], [R, 1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 1, 0]])$$

is the pre-Umbra

$$x_1^{A_1}x_2^{A_2}x_3^{A_3} \to q^{A_1+4}P_\infty()P_{A_1}(x_0)P_{A_2}(x_0)P_{A_3}(x_1, qx_2, x_3)P_\infty(qx_3) \quad .$$

while

$$APU(LLRR, [[C, C, [L, 0], [R, -1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 1, 0]])$$

is the pre-Umbra

$$\begin{aligned} x_1^{A_1}x_2^{A_2}x_3^{A_3} &\to q^{A_1+4}P_\infty()P_{A_1}(x_0)P_{A_2}(x_0)P_{A_3}(x_1, qx_2, x_3, qx_4)P_\infty() \\ &= q^{A_1+4}P_{A_3}(x_1, qx_2, x_3, q) \end{aligned}$$

and

$$APU(LLRR, [[C, C, [L, 1], [R, -1]], \{[1, 2]\}, [[0, 1], [2, 3], [3, 4], [4, 5]], [0, 0, 1, 0]])$$

is the pre-Umbra

$$x_1^{A_1}x_2^{A_2}x_3^{A_3} \to q^{A_1+4}P_\infty()P_{A_2}(x_0)P_{A_3}(qx_0,x_1,qx_2,x_3,qx_4)P_\infty()$$
$$= q^{A_1+4}P_{A_3}(q,x_1,qx_2,x_3,q).$$

At the end we also set $x_0=1$ and $x_4=1$.

Keeping Track of the Emerged Letter

Each of the many Followers induces a clear-cut SAP-letter to the right, so to keep track of it we will use yet another indexed variable, Z[EmergedLetter], and write, for example, the Atomic Pre-Umbra given above, as

$$x_1^{A_1}x_2^{A_2}x_3^{A_3}Z[LLRR] \to q^{A_1+4}P_{A_3}(q,x_1,qx_2,x_3,q)Z[LLRR] \quad .$$

(in this randomly chosen example the initial and final SAP letters coincided (both being LLRR), of course this is not generally the case).

At Long Last: The Preumbra

To get the action of the evolution Umbra on a generic monomial

$$x_1^{A_1}x_2^{A_2}\dots x_{2s-1}^{A_{2s-1}}Z[LETTER],$$

we first find all the Followers of $LETTER$, and for each and every one of them, we compute the Atomic Pre-Umbra, and finally add all these atomic pre-Umbras up.

Calling It Quits: How to End a SAP

We need one more letter, END. We can only end a SAP if the current letter is either LR, or LLRR, or LLRLRR, and in general $L(LR)^{s-1}R$, for $s \geq 1$. Then we close all the odd-numbered intervals, and call it a sap. Of course the function corresponding to END has 0 arguments, and the q-credit that it gets is $q^{A_1+A_3+\dots+A_{2s-1}}$. When this is the case we add $q^{A_1+A_3+\dots+A_{2s-1}}Z[END]$ to the above total.

The Pre-Umbral Matrix

Now we form a square matrix whose dimension is the size of the SAP alphabet (i.e. $C_1+C_2+\dots+C_r$), both of whose rows and columns are labelled by letters (legal parentheses with $\leq r$ LR pairs). The entry corresponding to the row labelled by $LETTER$ and column labelled by $LETTER'$ is the coefficient of $Z[LETTER']$ in the pre-umbra constructed above, which is our umbra applied to the generic monomial $x_1^{A_1}x_2^{A_2}\dots x_{2s-1}^{A_{2s-1}}Z[LETTER]$, where s is the number of LR-pairs of $LETTER$.

The Umbral Matrix

We now convert, as described in [Z1], each of the entries of the pre-Umbral matrix into a full-fledged umbra. (this is accomplished by procedure `ToUmbra` in `ROTA`, that has been transported to `USAP`).

The Umbral Scheme

Recall that ([Z1]) in addition to the Umbral matrix, an Umbral Scheme also needs a subset of *starting letters*, *ending letters*, and the *initial generating functions* for the starting letters. In this case things are easy. There is only one starting letter, START, one ending letter, END, and the initial vector consists of the unit vector with all 0's except for the component corresponding to START, that has a 1.

A User's Manual For The Maple package USAP

The reader is urged to study the source code of the Maple package `USAP` that for any *fixed*, but *arbitrary* positive integer r, *automatically* finds Umbral Schemes for SAPS with $\leq 2r$ horizontal edges per vertical cross-section. It does not require the Maple package `ROTA` (described in [Z1]), since all the necessary procedures of the latter were transported.

The main procedures of `USAP` are `SAPUmSc` and `ApplyUmSc`. If you want to get the Umbral Scheme for enumerating SAPs with $\leq 2r$ horizontal edges on each vertical cross-section, type `SAPUSr:=SAPUmSc(r,x,q);`. Here `r` is a specific positive integer (`r`$=1,2,\ldots$), while `x` is an indexed variable-name that will carry the lengths of the intervals between consecutive horizontal edges, and `q` is the variable of interest, that carries the perimeter. For example `SAPUS1:=SAPUmSc(1,x,q);` and `SAPUS2:=SAPUmSc(2,x,q);` give the Umbral Schemes for $r = 1$ and $r = 2$ respectively. Unfortunately, the case $r = 3$ would have to wait for a bigger computer and/or a more efficient implementation, since our computers ran out of memory. Since computing `SAPUS2:=SAPUmSc(2,x,q);` takes awhile, I decided to include the pre-computed SAPUS2. Thus typing `SAPUS2;` would display it. So see it in humanese, type: `YafeUmSc(SAPUS2,F,D);` , For the sake of completeness I also included `SAPUS1;` , even though it is instantaneous.

Once Maple gave you the Umbral Scheme (`SAPUS1` and `SAPUS2`), to get the first $n/2 - 1$ terms in the enumerating sequence (what physicists call "series expansions") type `ApplyUmSc(SAPUSr,q,n,var):`, where `var` is the *set* of catalytic variables:

$$\{x[1], x[2], \ldots, x[2 * r - 1]\}.$$

For example try,

```
ApplyUmSc(SAPUS1,q,20,{ x[1]}):
```

and

```
ApplyUmSc(SAPUS2,q,20,{ x[1],x[2], x[3] }): .
```

Typing `ApplyUmSc(SAPUS2,q,44,{ x[1],x[2],x[3] }):` and waiting for a few days computes the sequence of self-avoiding polygons with at most 4 horizontal edges per vertical cross-section to 21 terms (i.e. the number of such saps with perimeter up to 44). This sequence was not in Neil Sloane and Simon Plouffe's *Encyclopedia*[SP], and we reported it for inclusion in Sloane's *database*.

The sequence starts with:

$$1, 2, 7, 28, 124, 588, 2938, 15266, 81770, 448698, 2510813, 14277838, 82286365, 479610362,$$
$$2822332127, 16745262798, 100058822258, 601588809890, 3636591773529,$$
$$22088370875438, 134733261281038, \ldots$$

It fits between **M1779** (the sequence for ApplyUmSc(SAPUS1,q,n,{ x[1]}):, first introduced by Temperley[T]) and **M1780**, the exponentially difficult enumerating sequence for *all* saps.

It's Not My Fault That It is Sooo COMPLICATED, Even For a Computer

`USAP` can, *in principle*, find an Umbral Scheme for the class of saps with at most $2r$ horizontal edges per vertical cross-section, for any *fixed*, but *arbitrary* r. But the number of interfaces grows so fast with r that Maple ran out of memory even for the case $r = 3$. In other words, it refused to perform the command: `SAPUS3:=SAPUmSc(3,x,q);`.

I am almost sure that with a more careful implementation, and/or by transporting to numeric programming languages such as `C`, `SAPUS3:=SAPUmSc(3,x,q);` is still feasible, but frankly, I doubt whether it is worth the effort.

The most important point is that there *exists* a program that can generate (at least in principle) a polynomial-time (in n), scheme for enumerating saps with at most $2r$ horizontal edges per vertical cross-section, with perimeter $\leq n$, for any fixed given r. So what if the 'polynomial' is so big as to make it impractical? I will be very impressed if you can *just write* a program for *Satisfiability*, that takes $O(n^{1000000000000000})$ time and memory. I am not asking for sample output.

Self-Avoiding Walks

A *self-avoiding walk* (henceforth saw) on the square-lattice is a finite connected induced subgraph of the square lattice with two vertices of degree 1, and all the other vertices of degree 2. Of course we identify two saws that are translations of each other. Note that except for the length-0 saw, every one of our saws gives rise to two usual saws, since it can be walked in two opposite directions, and here we identify them, making a saw a *static* object.

The analogous program, *USAW*, for self-avoiding walks, is much more complicated, but goes along the same lines. Just as in [Z0], the SAW alphabet consists, in addition to the SAP alphabet, letters obtained from them by inserting one or two A's, denoting the "open ends". The three phases of following: PrePreFollower, PreFollower, and Follower, still hold, but there are many more possibilities, because of the open ends.

We urge the reader to carefully study the source code of `USAW`. We omit the humanese narrative, since the readers should be able to figure everything out by themselves. Note that `USAW` is an adaptation of `USAP`, and all the main procedures of the latter (except for the principal `SAWUmSc`) have the name of their `USAP` counterparts, with a "W" attached. For example, `FollowersW` is the SAW-adaptation of `USAP`'s `Followers`.

A User's Manual For The Maple package USAW

The main procedure of `USAW` is `SAWUmSc`. If you want to get the Umbral Scheme for enumerating SAWs with $\leq r$ L-R pairs on each vertical cross-section (and zero, one or two A's for the open ends) type `SAWUSr:=SAWUmSc(r,x,q);`. Here `r` is a specific non-negative integer (`r`$=0,1,2,\ldots$), while `x` is an indexed variable-name that will carry the lengths of the intervals between consecutive horizontal edges, and `q` is the variable of interest, that carries the length. For example `SAWUS0:=SAWUmSc(0,x,q);` gives the Umbral Scheme for enumerating initially-East-bound saws with at most one 'U-turn' in the East-West direction. This is a superset of the 'up-side' saws enumerated by Lauren Williams ([Wi]).

Typing `ApplyUmSc(SAWUS0,q,20,{x[1]}):` would give the first 20 terms, and multiplying all the terms, except the first by 2, gives the first 20 terms of the enumerating sequence for self-avoiding walks whose initial first horizontal edge is East-bound and whose projection on the x-axis performs at most one U-turn.

$$1,4,12,34,90,238,614,1584,4028,10246,25818,65076,\\ 162940,408074,1016962,2534848,6294268,15631572,38703172,95838918,\ldots.$$

You are welcome to try `SAWUS1:=SAWUmSc(1,x,q);` on *your* computer, but on *my* computer, it ran out of memory. So it seems that `SAWUS2:=SAWUmSc(2,x,q);` would be *really* hopeless, but so what? Once again the *program is the message*, not the output. For any given r I have an *effective* program that finds a polynomial-time scheme for enumerating saws with up to r LR-pairs. And sometimes, having the program is our only substitute to seeing the output. I would be very impressed if you can *write* a program, that in finite time and memory, is *guaranteed* to output `true` or `false`, depending on whether the Riemann Hypothesis is true or false, I am not asking you to show me the output.

MAYLIS: a Targeted Package for Generalizing the case $r = 1$ of USAP

The case $r = 1$ of `SAPUmSc` corresponds to enumerating *column-convex* polyominoes (lattice animals) by perimeter, solved implicitly by Temperley, and completed by Brak, Guttmann, and Enting ([BGE]). `MAYLIS` is capable of not only doing the Temperley part (that is already done in `USAP`) but also the part done in [BGE].

All you have to do is type

$$Xavier(\{[-1,-1],[-1,0],[-1,1],[0,-1],[0,0],[0,1],[1,-1],[1,0],[1,1]\},q); \quad .$$

`SSUE` automatically tries to solve the equations implied by the Umbral scheme that turn out to have the form

$$A(x,q)F(x,q)+B(x,q)F(q,q)+C(x,q)\frac{\partial F}{\partial x}(q,q)+D(x,q)=0 \quad , \qquad (Doron)$$

for some *polynomials* A, B, C, D in (x, q). In the fortunate case, plugging in $x = q$ and then differentiating with respect to x, and then plugging $x = q$, produces two linear equations for the two unknowns $F(q, q)$ and $\frac{\partial F}{\partial x}(q, q)$, that must be rational functions in this lucky case. $F(q, q)$ is the desired generating function. In the unfortunate case, plugging $x = q$ produces the tautology $0 = 0$. In this case we can solve the algebraic equation $A(x(q), q) = 0$, pick those $x(q)$ that define a formal power-series, and plug into $(Doron)$ and its derivative w.r.t. x. We obtain linear equations for the two unknowns $F(q, q)$ and $\frac{\partial F}{\partial x}(q, q)$ that can be solved, yielding this time *algebraic* generating functions. Because of what Gregory Chaitin calls *programmer's fatigue*, I only treated the cases where $A(x, q)$ is either linear, quadratic, or a product of these, but two more hours of programming would have done the general case.

`Xavier` can do many more cases, just as fast (and often faster). The first argument to procedure `Xavier` is a set of "allowed interfaces" between one vertical cross-section to the next. We use the same conventions as in *PreFollowers*. '1' means up, '0' means straight ahead, and '-1' means down. For example, the interface [1,1] means both the (sole) L and the (sole) R go up, [0,0], means they both continue horizontally, etc. there turned out to be 271 inequivalent cases to consider (about half of 2^9, the number of subsets of all the nine possible interfaces). For example try `Xavier(`$\{[0,0]\}$`,q);` for the trivial case of enumerating rectangles, and `Xavier(`$\{[0,0],[1,0]\}$`,q);` for enumerating Ferrers diagrams by perimeter. Surprisingly, the generating function for 'strictly-upwards' convex polygons, gotten by typing `Xavier(`$\{[1,1]\}$`,q);` gives the generating function for Generalized Catalan Numbers **M1141** of Sloane-Plouffe[SP], that enumerates secondary structures of RNA ([Wa]).

To see all the successful outputs of `Xavier`, type: `DoThemAll(q);` . With the current version of `SSUE`, Ekhad succeeded in doing 81 cases and failed in the remaining 190, but I am sure that an improved version of `SSUE` would give a %100 success rate.

Variety is the spice of life. An $r = 1$ sap may first follow one set of interfaces, then another, then yet another, and so on. So every *word* in the alphabet whose letters are all the 511 non-empty subsets of the set of nine interfaces $\{0, -1, 1\}^2$, introduces yet another combinatorial family. This is computed, automatically, of course, by procedure `Delest1` (if you are not allowed to quit), or `Delest` (if you are allowed to quit).

It is easy to see that a *convex* polygon has three parts, first 'weakly outwards' with allowed interfaces $\{[-1, 0], [-1, 1], [0, 0], [0, 1]\}$, then 'weakly upwards' with allowed interfaces $\{[0, 0], [0, 1], [1, 0], [1, 1]\}$ (or 'weakly downwards' with the negative of the above), then 'weakly inwards', with allowed interfaces $\{[0, 0], [0, -1], [1, 0], [1, -1]\}]$. The generating function enumerating the weakly upwards convex polyominoes, where you are also allowed to quit in the middle is gotten by typing

$$Delest([\{[-1,0],[-1,1],[0,0],[0,1]\},\{[0,0],[0,1],[1,0],[1,1]\},\\ \{[0,0],[0,-1],[1,0],[1,-1]\}],q);$$

and the generating function for convex polyominoes by perimeter is roughly twice that. To get the exact count, we have to account for some trivial over-counting. This is done in `ProveDelestViennot();` , that, as we already boasted above, proves the Delest-Viennot formula in about 10 seconds of CPU time.

Multi-Parameter Counting

`MAYLIS` also contains procedures to find Umbral Schemes for counting saps (for arbitrary r, not just $r = 1$), according to *area* and *width* as well as *perimeter*. It is procedure `SAPUmScg` whose syntax is: `SAPUmScg(r,q,x,t,w);` . It can also handle restricted interfaces with `SAPUmScgR`. See the on-line help by typing `ezra(SAPUmScgR);` .

To Do List

1. Extend `MAYLIS`'s procedure `SSUE` to solve more general equations, that also involve the $q-$ dilation operation $f(x) \to f(qx)$, thereby finding computer-generated proofs to the amazing results summarized in [B], for enumerating various families of convex polyominoes according to *area, width* and *perimeter*.

2. Use the methodology of [Z3] and [Z4] (this paper), to study 'toy models' for the Ising model with magnetic field, Percolation, and other venerable models of statistical physics.

3. One of the applications of an Umbral Scheme is to actually *crank out* numbers, obtaining *series expansions* for the toy-model, that give lower bounds for the 'real thing'. For that application, it seems wasteful to have the full Umbral Scheme, since we only need to know the first prescribed terms of the Taylor expansions of the coefficients that feature in the scheme. Adapt procedures `SAPUmSc` of `USAP` and `SAWUmSc` of `USAW` to find 'approximate' schemes that will take advantage of the above observation, and produce many more terms before running out memory.

4. Translate everything to Fortran or C, hopefully getting much more impressive output.

5. Physicists are most interested in *critical exponents*. Find an algorithm that inputs an Umbral Scheme, and outputs the critical exponent for the desired generating function.

Appendix: The Umbral Scheme For Saps With At Most 2 Horizontal Edges Per Vertical Cross-Section

The desired quantity is $F_3()$.

$$F_1() = 1,$$

$$F_2(x) = -\frac{F_1()q^3x}{-1+qx_1} + \frac{F_2'(q)q^2x}{-x+q} + \frac{F_2(x)q^2x^2(q-1)^2(q+1)^2}{(-x+q)^2(-1+qx)^2} - \frac{F_2(q)q^2x^2(-qx+2\,q^2-1)}{(-x+q)^2(-1+qx)},$$

$$F_3() = F_2(q).$$

The corresponding much larger output for at most 4 horizontal edges can be seen on the home page of Zeilberger: `http://www.math.temple.edu/~zeilberg/` .

REFERENCES

[B] Mireille Bousquet-Mélou, *Rapport Scientifique pour obtenir l'Habilation à diriger des Recherches*, available from `http://dept-info.labri.u-bordeaux.fr/~bousquet/`.

[BGE] R. Brak, A. J. Guttmann, and I. G. Enting, *Exact solutions of the row-convex polygon perimeter generating function*, J. Phys. A: Math. Gen., **23** (1990), 2319-2326.

[DV] M. -P. Delest and G. Viennot, *Algebraic languages and polyomino enumeration*, Theoretical Computer Science **34** (1984), 169-206.

[SP] N. J. A. Sloane and S. Plouffe, "*The Encyclopedia of Integer Sequences*", Academic Press, 1995.

[T] H. N. V. Temperley, *Combinatorial problems suggested by the statistical mechanics of domains and rubber-like molecules*, Physical Review **103** (1956), 1-16.

[Wa] M. S. Waterman, *Secondary structures of single-stranded nucleic acids*, in: "Studies in the Foundation of Combinatorics", v. 1, 167-212, 1978.

[Wi] Lauren K. Williams, *Enumerating up-side self-avoiding walks on integer lattices* (10pp), Electronic J. Combinatorics **3** (1996), R31.

[Z0] Doron Zeilberger, *Symbol-Crunching with the Transfer-Matrix Method in Order to Count Skinny Physical Creatures*, INTEGERS (`http://www.integers-ejcnt.org`), **0** A9 (29 pages) (2000).

[Z1] Doron Zeilberger, *The Umbral Transfer-Matrix Method. I. Foundations*, J. Comb. Theory Ser. A **91** (2000), 451-463. [Rota memorial issue].

[Z2] Doron Zeilberger, *The Umbral Transfer-Matrix Method. II. Counting Plane Partitions*, Personal Journal of Ekhad and Zeilberger,
`http://www.math.temple.edu/~zeilberg/pj.html`.

[Z3] Doron Zeilberger, *The Umbral Transfer-Matrix Method. III. Counting Animals*, submitted. Available from `http://www.math.temple.edu/~zeilberg/papers1.html`.

[Z4] Doron Zeilberger, *The Umbral Transfer-Matrix Method. IV. Counting Self-Avoiding Polygons and Walks*, this article.

[Z5] Doron Zeilberger, *The Umbral Transfer-Matrix Method. V. The Goulden-Jackson Cluster Method for Infinitely Many Mistakes*, in preparation.